Roadrunner and the Future of Applications Programming

Paul Henning

Roadrunner Project/CCS-2
Los Alamos National Laboratory
phenning@lanl.gov

July 8, 2008

LA-UR 08-04400
With many apologies to Edgar Allan Poe…

But the Roadrunner still beguiling my sad fancy into smiling,
Straight I wheeled a cushioned seat in front of bird and bust and door;
Then, upon the velvet sinking, I betook myself to linking
Fancy unto fancy, thinking what this ominous bird of yore -
What this grim, ungainly, ghastly, gaunt, and ominous bird of yore
Meant in croaking ‘Nevermore.’
Is applications programming as we know it ‘Nevermore?’

- A brief survey of contemporary architectures
  - Roadrunner overview
  - New systems
- Some programming considerations for Roadrunner and beyond
- A possible (although steep, rocky, and treacherous) path forward
A brief look at Roadrunner

http://www.lanl.gov/roadrunner
A Roadrunner “Triblade” node integrates Cell and Opteron blades

- **LS21** is an IBM dual-socket Opteron blade
- **QS22** is an IBM Cell blade containing two PowerXCell 8i™ enhanced double precision CBES
- Expansion blade connects two QS22 to LS21 via four PCI-E x8 links and provides the node’s single Infiniband 4X DDR cluster attachment
- Roadrunner Triblades are completely diskless
- Roadrunner is a cluster-of-cluster of Triblades: 180 nodes x 17 CUs
Roadrunner embodies most probable features of near-term systems

- Cluster computing (MPI)
- Multicore SMP processors (Opterons, Cells)
- 44 threads per node (4 Opteron, 8 PPE, 32 SPE), ~135K total
- 3 distinct address spaces per node (one per blade)
- Heterogeneous instruction sets (x86, PowerPC, SPU)
- SIMD floating point (SSE, AltiVec, SPU)
- Local stores instead of caches (SPU)
- On-chip CPU/memory networks (Cell EIB)
- “Remote” accelerators (Cell blades on PCI-E)
- 586 Mflops/Watt
New processors will make their way into next-generation supercomputers

NVIDIA “GTX 280”
240 SPs

Sun “Rock”
16-64 threads

Intel “Larrabee”

AMD “Fusion”

Taken from publicly available information

Los Alamos National Laboratory
EST. 1943
Operated by Los Alamos National Security, LLC for NNSA
... but those products already “exist.” What’s next?

- As much as I can say is:
  - Core counts are increasing quickly
  - More specialized heterogeneous cores are appearing on chip
  - More “interesting” cache/memory/local store layouts
  - Some chips are specialized for HPC
  - SIMD lengths are growing

- Double-precision teraflop (-ish) chips will be here soon
  - Petaflop is obtainable in a medium-sized cluster!
Applications programming considerations
The simplest Roadrunner programming model is one Cell chip per Opteron core.
A variety of application models have been used

- **Cell-centric**
  - Application primarily on the Cell processors
  - Opterons used for communication, I/O and collective operations
  - Issues: lack of PPE speed and size of SPE local store
  - Good for new compact codes

- **Opteron-centric**
  - Application primarily on the Opteron processors
  - Cells used to accelerate numerically-intensive regions
  - Issues: data transfer times from Opteron to Cell, not wasting the Opteron cycles
  - Good for accelerating existing codes

- **Decomposed**
  - Functionally-decompose application across all units
A Cell-only tuning example can be found in SPE DMAs

- The Cell SPE processors need to transfer data from Cell main memory (4 GiB) to SPE local store (256 KiB)
- SPE has a dedicated DMA processor (the “MFC”) for true asynchronous transfers: 16 in-flight per SPE, ≤16 KiB per transfer
- All processors (8 SPEs, PPE L2 cache misses) access RAM across the 25.6 GiB/s Element Interface Bus (4 unidirectional rings)
- Performance questions:
  - How many buffers should an SPE have (e.g. inbound, compute, outbound) to cover communication with compute?
  - How big should any one transfer be?
  - How many SPEs should be working?
SPE DMA performance tuning

Time required to complete a large DAXPY varying the number of SPEs (1-6,8) and the number of communication buffers (1-6).
Roadrunner makes this a little more difficult

- Another level of memory transfer: Opteron to Cell
- Where does SPE data partitioning occur?
- Where does byte-swapping occur?
- How do you overlap communication and computation between all three levels of processor?
- Do you optimize data structures for Opteron or for Cell?
That was enough easy stuff!
Major application development issues are looming

- Explosion in platform-specific techniques, directives, tools & protocols
- Changes in SIMD length can have significant data structure implications
- MPMD, even for homogeneous systems, means code base growth
- Debugging (and profiling) becomes more and more difficult
  - No common clock (or even clock frequency), even on-node
  - Multiple instruction sets - what does “step” mean?
  - Massive thread/process counts
  - Non-standard interconnects in accelerator-based systems - data acquisition
- Heterogeneous instruction sets amplify all of the above
- Applications need to be aware of power management, reliability, I/O imbalance, etc.
  - Google: failure-tolerant applications on failure-prone hardware
- More important to optimize data motion than instruction counts!
Existing production software tools are part of the solution...

- “Production” implies Fortran/C/C++ plus MPI and/or pthreads
- Compilers are reasonable at transforming imperative programs to assembly code, for a particular ISA
- Libraries (+autotuning) provide cross-platform functionality and spread the knowledge of experts over a large user base
- A lot of work has gone into other parts of the toolchain
... as are “new” developments...

- Applying autotuning to applications
- PGAS
- Domain calculus
- Higher-level modeling of concurrency
- Modern HPC languages
- Multicore libraries (IBM ALF, Intel Ct, Microsoft TPL, etc.)
- Accelerator languages (CUDA, Stream SDK, OpenCL, etc.)
- But…
... but there are still a lot of problems.

- Libraries have limited applicability and impose data motion/translation issues
- Can PGAS handle deep, disjoint memory hierarchies?
- Most new languages push hard parallelism issues to developer
- SPMD is certainly dead at the ISA level, probably at the source level
- Compute-intensive regions often require large data tables
- The new languages are still imperative (although less deterministic)
  - Makes high-level analysis difficult
  - Obfuscates the intent of the code
  - Makes the developers select data structures, one of the hardest things for compilers to change
- Multiple communication protocols may be needed
Obvious observations *should* guide our work

- Scientific computing is more about results than applications
- Physics and methods people are easier to fund than CS people on “science” projects
- HPC applications are requiring ever more machine-specific design considerations (compare “high-performance” vs. “large scale”)
- Many computational patterns are re-used in scientific computing (refinement of the \{7,13\}-dwarfs):
  - Sweeps over mesh features
  - Treating a field variable as linear systems or FFTs
  - Tracking particles
  - Data table lookups
  - n-Body calculations
Scientific code generators may provide a path forward

- Provide a high-level language for describing calculations (e.g. coupled multiphysics simulations on meshes)
- Provide interfaces for packaging solvers and data
- Autonomous or guided selection of patterns for \{calculations\} \times \{machine features\}
- Generate source code targeted for a machine, or even a particular set of inputs
- Leverage existing tool sets
  - Compilers
  - Communications
  - Build tools
  - Run scripts
The benefits of decoupling the “what” from the “how” are numerous

- Separates domains of expertise
- Preserves institutional knowledge
- Enables rapid experimentation with methods
- Provides opportunity for machine-selected data structures
- Allows performance gains through problem-specific specialization
- Simplifies V&V and cross-platform comparison
- Accelerates application deployment on new architectures
- Allows domain-specific error-checking (e.g. unit analysis)
We will need to use every tool at our disposal
“Nevermore?” I hope so, but it won’t be easy

- Roadrunner provides a balance of new technologies from which to explore the future (and it is fun!)
- It is difficult to use all the features of all of the new systems if you have describe how to do your work on each of them.
- SPMD imperative programming gets the job done. Let’s not forget that.
- SPMD imperative programming is error-prone, poorly-coupled, system-at-a-time high-level assembly. Let’s move on.
- By limiting the domain of applicability and raising the level of abstraction, we can get computers to do more of tedious bookkeeping work required for generating imperative programs.
- This audience has the expertise required to get the job!
Gedankenexperiment: port a particle code

1. You need to represent a time-varying number of particles, each with position, direction, etc. moving through (and interacting with) physics quantities stored in a background mesh

2. Pick a data layout, assuming a 128-bit SIMD general purpose processor
   ■ Did you get your intra- and inter-structure data alignment correct?

3. Communicate your data (particle & mesh) to another address space
   ■ Of course, you didn’t waste cycles copying/serializing your data or dereferencing pointers

4. Partition your work among \( N >> 16 \) specialized processors, \( -O(\infty) \)
   ■ Is your data structure optimal for these processors?

5. Now make this code efficient on a 1024b SIMD processor from another vendor in another interconnect topology